

Ideas of VisComposer/梅鸿辉

期望结果

1. 比较完善严谨的 Scenegraph 定义，拥有足够的表达能力
2. 迭代地增量式修改，尽可能减少其中每一步的理解和操作难度，同时能交替地在数据处理（选择）和视觉绑定之间任意切换；能很好地辅助用户将问题进行拆分和重组。
3. 背后有一定的理论模型支撑

Idea : 结构化 Scenegraph

以各种结构（分支、循环、递归）从另一个角度定义 Scenegraph，以期获得比简单树状结构更加灵活的表达能力

常见的工具（D3、VEGA）基于的场景树是纯树状结构，每个父节点拥有若干子节点，实际相当于一个循环（D3 的 `selection.data` 通常需要传入一个数组），此外还可以借鉴程序设计的控制流（Control Flow）设置更多结构：

- 线性

最简单的结构，手动指定每个节点。相当于 D3 中的 `selection.datum`

- 循环

最常见的结构，将数据按顺序分发给子节点。相当于 D3 中的 `selection.data`，其关键组成部分包括数据、数据长度、索引和分发方式

■ 一般来说是确定运行次数循环（for），可以考虑是否需要添加条件判断循环（while）

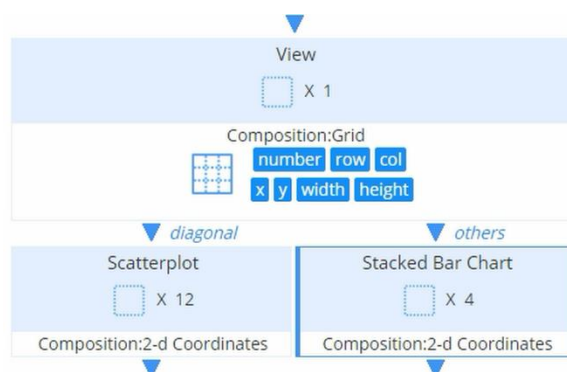
- 分支（条件判断）

如右图所示，树在某个节点位置分裂为两个或者更多节点。

难点

1. 分支发生在数据分派（即场景树的边）之前还是之后？
2. 分支后的节点未必是完全不同的两个节点，很多时候他们需要共享一部分属性
3. 分支节点可以拥有共同的子节点，这样使得场景图真的从“树”变为“图”；为了实现这一目的，需要两个父节点能提供给予子节点相同的数据接口

- 递归



递归结构（如 Treemap）是目前可视化构建工具都不能很好解决的内容，其实现方法主要有两种：

1. 通过 layout 函数将其平面化（如 Treemap 被变换为一系列矩形），绘制出的 DOM 为单层结构并非多层的递归结构。
2. 自行写代码实现递归；这样就脱离并打破了提供的框架

难点

1. 类似分支中的难点 3，如何定义标准化的接口进行递归转移；此外还要考虑递归终止条件
2. 增加了 Scenegraph 的理解难度和调试难度

与之前（D3）的区别

事实上，除递归外，都是 D3 中的 scenegraph 可以支持的内容（递归可以靠外部代码实现，但这样就一定程度上脱离了其本身框架结构）。区别在于 D3 是一种自顶向下的设定方式（宽度优先），后者可以是另一种顺序（深度优先）。

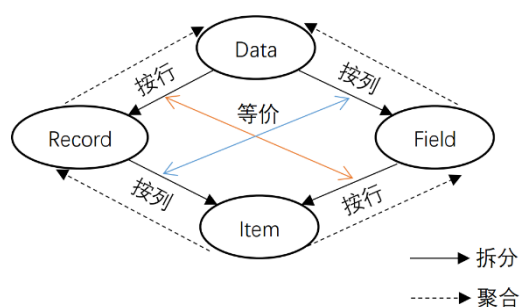
D3 的代码（包括 VEGA）倾向于将数据全部组织好后再开始逐层设置，于是大部分的布局都通过一个 layout 函数将原始数据计算成整理好的格式，再进行绘制。对于用户来说，因为最初不清楚整理后的数据到底以何种形式排列，所以上手时只能先行复制样例再进行修改。

难点

1. Scenegraph 结构变复杂，需要考虑如何通过连线、符号、空间位置等对其进行清晰的表达
2. 设计一种数据结构序列化复杂的 Scenegraph 结构用于模板保存

Idea：状态机

Task=交互>>新节点：将各种任务（filter、aggregate 等）看做在某个数据和视图进行特定交互后生成的数据和视图（即新的 Scenegraph 节点）；通过一个状态机模型指导新节点的生成结果，例如对于表格数据如右图所示：对数据进行的操作（按行选取、按列选取、聚合等）作为转移



可能的应用

1. 提供半自动的节点生成
2. 生成最小生成树以减少重复计算、提高效率

难点

1. 如果自动生成状态机，如何定义状态（牵扯到一致性的判定，两个状态是否是同一个）和转移（如何通过一个交互自动生成转移）

-
2. 是否要对用户显示？如果显示如何让用户理解？
 3. 同时存在多种数据源时的融合问题

存在的问题

如果这个模型不是非常复杂的话就没有必要设计状态机来完成, 但很难找到如此复杂的使用场景。

Idea : Construction for Novices

以 How information visualization novices construct visualizations[1]这篇文章中提出的一系列难点、新手希望进行的可视化构建步骤作为理论基础, 通过 Scenegraph 和相关功能提供相应设计流程。

其中, 将可视设计氛围了三个步骤: 数据属性选择、视觉模板选择和视觉映射指定。新手会在这三种模式间来回切换。

而新手经常尝试进行 (但工具不支持) 的操作其中有几条:

- See without mapping them to any visual property

参考下一个 Idea, 在一定程度上将系统作为 dataflow 的处理和展示而非视图的定制, 同时为处理了的数据流提供一系列默认的查看方式。

- Partial specification

我觉得是很能体现 Scenegraph“先搭构架, 再实现细节”的设计思路的地方。

难点

1. 在节点没有完成全部视觉绑定前如何进行占位显示
2. 在整体结构部分缺失时提供在局部较为完善的操作和查看

实现来说, 可以考虑参照上一点, 先进行一些数据处理 (过滤等) 选中其中一部分子数据进行可视设计, 然后回头补齐得到完整结果。子数据和其对应视图并入整体时通过上文状态机确定合并方式 (决定作为哪个节点的子节点等)

Idea : 数据流处理

VisFlow[2]中使用了在自由 canvas 中拖拽拉线的方式进行 dataflow 的构建，但其重点在于数据的处理查看而非视图的构建。

本工作并非着重于数据处理，但可以考虑类似的操作方式，但目的不同：

- 如前所述，D3、VEGA 和 ECharts 等倾向于将数据全部组织好后再开始逐层设置，其数据域经常是一些特定复杂结构的数据，而让用户在开始具体的视觉设计前就组织好数据结构是有难度的（如右图是 ECharts 中分组柱状图使用的数据格式，一般的源数据文件都不会使用这样的格式）。
- 应当将数据处理（选择）的步骤与可视设计步骤融合，可以通过上述 Partial specification 和状态机模型实现。

```
xAxis : [
  {
    type : 'category',
    data : ['周一', '周二', '周三', '周四', '周五', '周六', '周日']
  }
],
yAxis : [
  {
    type : 'value'
  }
],
series : [
  {
    name: '直接访问',
    type: 'bar',
    data: [320, 332, 301, 334, 390, 330, 320]
  },
  {
    name: '邮件营销',
    type: 'bar',
    stack: '广告',
    data: [120, 132, 101, 134, 90, 230, 210]
  },
  {
    name: '联盟广告',
    type: 'bar',
    stack: '广告',
    data: [220, 182, 191, 234, 290, 330, 310]
  },
  {
    name: '视频广告',
    type: 'bar',
    stack: '广告',
    data: [150, 232, 201, 154, 190, 330, 410]
  }
],
```

难点

1. 将数据处理与视觉设计结合而不至于喧宾夺主
2. 如何以交互式地进行而不产生大量复杂操作

Idea : 历史记录

保存树状的操作记录，提供撤销、重复和合并操作（类似 git）。可以看做另一种形式的模板存储、复用。

Idea : 交互辅助

注重 direct manipulation 的增强。在 PS 等软件中，会需要先选取图层然后再进行操作，以明确操作对象。Scenagraph 中的节点可以起到与 PS 中图层类似的效果，明确需要操作的视图和数据，解决数据选择和视觉映射上的一些问题。

Idea : 动画和交互的设计

以 D3 为例，一个对象拥有 enter、update 和 exit 几种生命周期，以及 on 事件响应，在这些节点上定义变换。由于是基于过程的变换，并不保证其完整性、可逆性。

我们考虑通过定义对象的不同状态（生命周期）和状态间变换（事件响应）进行交互和动画的设计。不同状态通过上述分支结构定义，状态间转换则涉及上述 Partial specification（仅

部分属性变化), 同时需要一定的交互设计减少其操作复杂度。

引用

- [1] L. Grammel, M. Tory, and M. A. Storey, "How information visualization novices construct visualizations," *IEEE TVCG*, vol. 16, no. 6, pp. 943–952, 2010.
- [2] B. Yu and C. Silva, "VisFlow - Web-based Visualization Framework for Tabular Data with a Subset Flow Model," *IEEE TVCG*, vol. 2626, no. August, pp. 1–1, 2016.